

Język PL/SQL

- Rozkazy języka SQL są niewystarczające do tworzenia efektywnych systemów baz danych – kontrola warunków integralności danych
- Firma Oracle wprowadza rozszerzenia o elementy programowania proceduralnego i obiektowego dla swojej implementacji języka SQL, które nazywa językiem PL/SQL – (tylko w SZBD Oracle)
- PL/SQL umożliwia definiowanie
 - anonimowych bloków programowych
 - procedur i funkcji składowanych w bazie danych
 - pakietów (bibliotek) procedur i funkcji
- W PL/SQL nie można umieszczać instrukcji DDL i DCL

Język PL/SQL - korzyści

- prostota wykonania niektórych zadań w stosunku do SQL
- większa wydajność
- dostępność nieobecnych w SQL mechanizmów
 - stałe, zmienne
 - struktury sterujące
 - obsługa wyjątków
- przenaszalność pomiędzy platformami, na które oferowany jest Oracle (kod wykonywany na serwerze)
- możliwość wykorzystania predefiniowanych pakietów

Typy danych w PL/SQL

Typy liczbowe

BINARY_INTEGER
DEC
DECIMAL
DOUBLE PRECISION
FLOAT
INT
INTEGER
NATURAL
NATURALN (not null)
NUMBER
NUMERIC
PLS_INTEGER
POSITIVE
POSITIVEN (not null)
REAL
SIGNTYPE
SMALLINT

Typy znakowe

CHAR
CHARACTER
LONG
NCHAR
NVARCHAR2
RAW
STRING
VARCHAR
VARCHAR2

Typ logiczny

BOOLEAN, literały:
TRUE (prawda),
FALSE (fałsz)

Typy czasowe

DATE
TIMESTAMP
INTERVAL

Typy złożone

RECORD
TABLE
VARRAY

Typy wskaźnikowe

REF CURSOR
REF object_type

Wykład : Wybrane typy danych

W Oracle jest bardzo wiele typów danych

Typ numeryczny	
Number(P,S)	może przechowywać typy rzeczywiste oraz całkowite. P oznacza ilość cyfr w całej liczbie, natomiast S oznacza ilość miejsc po przecinku
Typy znakowe	
Char(L)	Przechowuje stałą ilość znaków zadeklarowaną jako parametr L. L musi być mniejsze niż 256. Przykładowo jeśli do kolumny typu Char(100) wstawimy pojedynczą literę, przechowywane będzie 100 znaków. Wartość zostanie uzupełniona do 100 spacjami.

Wykład : Wybrane typy danych

Typy znakowe	
Varchar2(L)	<p>Dane tego typu zajmują dokładnie tyle miejsca ile zostanie przypisanych znaków. Parametr L określa maksymalną ilość znaków. L musi być mniejsze od 4000.</p> <p>Ze względu na optymalizację, dane tekstowe zazwyczaj przechowuje się w typie Varchar2. W użytku codziennym używa się typu Char raczej sporadycznie i tylko wtedy gdy zachodzi taka konieczność.</p>
Long	<p>Posiada własności takie jak VARCHAR2, jednak może przechowywać do 2 GB tekstu.</p>

Wykład : Wybrane typy danych

Inne typy	
Raw(L)	Typ zachowujący się jak CHAR(L), jednak służy do przechowywania danych binarnych.
Long raw	Odpowiednik typu Long, tylko do przechowywania danych binarnych.
Date	Przechowuje informacje o wieku, roku, miesiącu, dniu, godzinie, minucie oraz sekundzie.
Timestamp(L)	Typ zbliżony do Date. Przechowuje dodatkowo do L miejsc po przecinku sekundy. L nie może być większe niż 9.
BLOB, CLOB, BFILE	Służą do przechowywania bardzo dużych plików. Ze względu na ich wysoką wydajność zaleca się korzystanie z tych typów zamiast typów LONG i LONG RAW.

Anonimowy bloków PL/SQL

Podstawowa jednostka programowa PL/SQL

[**DECLARE**

-- sekcja deklaracji]

BEGIN

-- instrukcje

[**EXCEPTION**

-- obsługa wyjątków]

END;

Bloki mogą tworzyć strukturę zagnieżdżoną

Anonimowy bloków PL/SQL

postać podstawowa

BEGIN

-- instrukcje programu

END;

postać pełna

DECLARE

-- sekcja deklaracji

BEGIN

-- instrukcje programu

EXCEPTION

-- obsługa wyjątków

END;

Zmienna

- Zmienne służą do przechowywania wyników zapytań i obliczeń w celu ich późniejszego wykorzystania
- Wszystkie zmienne muszą być zadeklarowane przed ich użyciem
- Każda zmienna posiada typ - takie same typy jak w SQL

Zmienna

- Zmienne deklaruje się w sekcji DECLARE
- Mamy zmienne:
 - **proste** – liczba, łańcuch znaków, data, wartość logiczna
 - **złożone** – rekord, tablica obiekt
- Dostępne w bloku deklaracji i blokach zagnieżdżonych

Przykłady
deklaracji zmiennych

```
DECLARE  
test NUMBER(6);  
nazwa VARCHAR2(100);  
data DATE;  
obecna BOOLEAN;
```

Zainicjalizowanie zmiennej

- Zmienna niezainicjalizowana ma wartość pustą
- Sposoby inicjalizowania zmiennej:
 - **przypisanie wartości** (:= operator przypisania)
 - **określenie wartości domyślnej** – (DEFAULT)
- Można wymusić obowiązkowość wartości (NOT NULL)

DECLARE

```
ocena NUMBER(6) NOT NULL:= 5;  
nazwa VARCHAR2(100) := 'ZAAWANSOWANE';  
data DATE DEFAULT DATE '2014-02-24';  
egzamin BOOLEAN NOT NULL DEFAULT TRUE;
```

Zmienna rekordowa

- PL/SQL posiada zmienne strukturalne nazywane rekordami - podzielone są na pola, z których każde posiada nazwę i typ
- Sposób deklaracji zmiennej rekordowej:
 - zdefiniowanie typu rekordowego
 - zdefiniowanie zmiennej typu rekordowego
- W programie używane przez dostęp kropkowy

definiowanie
typu rekordowego

```
TYPE DaneOsobowe IS RECORD (  
  nazwisko VARCHAR2(25),  
  imie VARCHAR2(20));
```

Przykład - zmienna rekordowa

DECLARE

TYPE DaneOsobowe **IS RECORD** (

nazwisko VARCHAR2(25),

imie VARCHAR2(20));

osoba DaneOsobowe;

BEGIN

osoba.nazwisko := 'Iksiński';

osoba.imie := 'Alfred';

Atrybuty obiektów

- Wszystkie obiekty posiadają atrybuty
- Atrybut **%TYPE** służy do deklaracji zmiennej prostej na podstawie typu atrybutu tabeli bazy danych lub typu innej zmiennej
- Atrybut **%ROWTYPE** służy do deklaracji zmiennej rekordowej na podstawie schematu tabeli bazy danych, kursora lub typu innej zmiennej rekordowej
- Najczęściej atrybutu **%ROWTYPE** używa się, gdy potrzebujemy zmiennej rekordowej, która ma przechować cały rekord ze wskazanej relacji

Stałe

- deklaracja w sekcji **DECLARE**
- musi być przy deklaracji zainicjalizowana
- nigdy nie zmienia przypisanej wartości

DECLARE

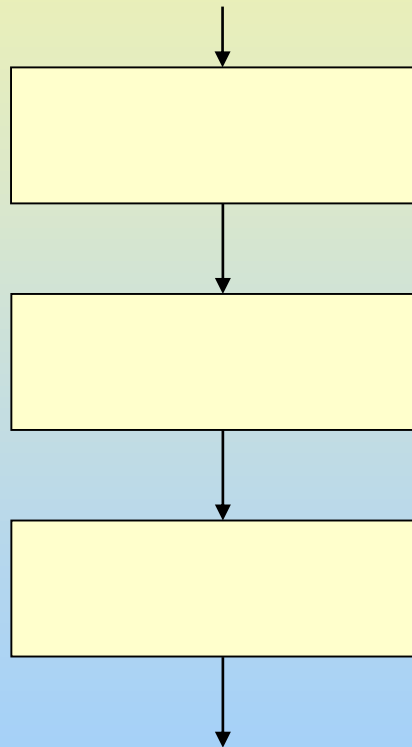
nazwa_zmiennej **CONSTANT** typ(długość) := wartość;

przykład inicjalizacji stałych

DECLARE

pi CONSTANT NUMBER(5,4) := 3.1415;
prawda CONSTANT BOOLEAN := TRUE;

Rodzaje struktur sterujących ***SEKWENCJA***



SEKWENCJA

- Ciąg poleceń wykonywany w określonym porządku
- umieszczana jest w sekcjach:
 - wykonywalnej
 - obsługi błędów bloku PL/SQL

DECLARE

```
test NUMBER(3) := 0;  
nazwa VARCHAR2(30);
```

BEGIN

```
test := test + 1;  
nazwa := 'Zaawansowane technologie '  
nazwa := nazwa || 'baz danych';
```

END;

Komunikacja z użytkownikiem

- Pobieranie informacji od użytkownika

```
zmienna := &zmienna_podstawienia;
```

- Wypisanie informacji na konsoli – pakiet DBMS_OUTPUT (procedura PUT_LINE)

```
DBMS_OUTPUT.PUT_LINE(ciąg_tekstowy);
```

- Przed wykonaniem programu trzeba ustawić w SQL*PLUS wartość zmiennej SETSERVEROUTPUT na ON

```
SET SERVEROUTPUT ON
```

komunikacja z użytkownikiem

- Przykład

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
i NUMBER(3) := &liczba;
```

```
nazwa VARCHAR2(50) := '&tekst';
```

```
BEGIN
```

```
dbms_output.put_line('Zmienna i: ' || to_char(i));
```

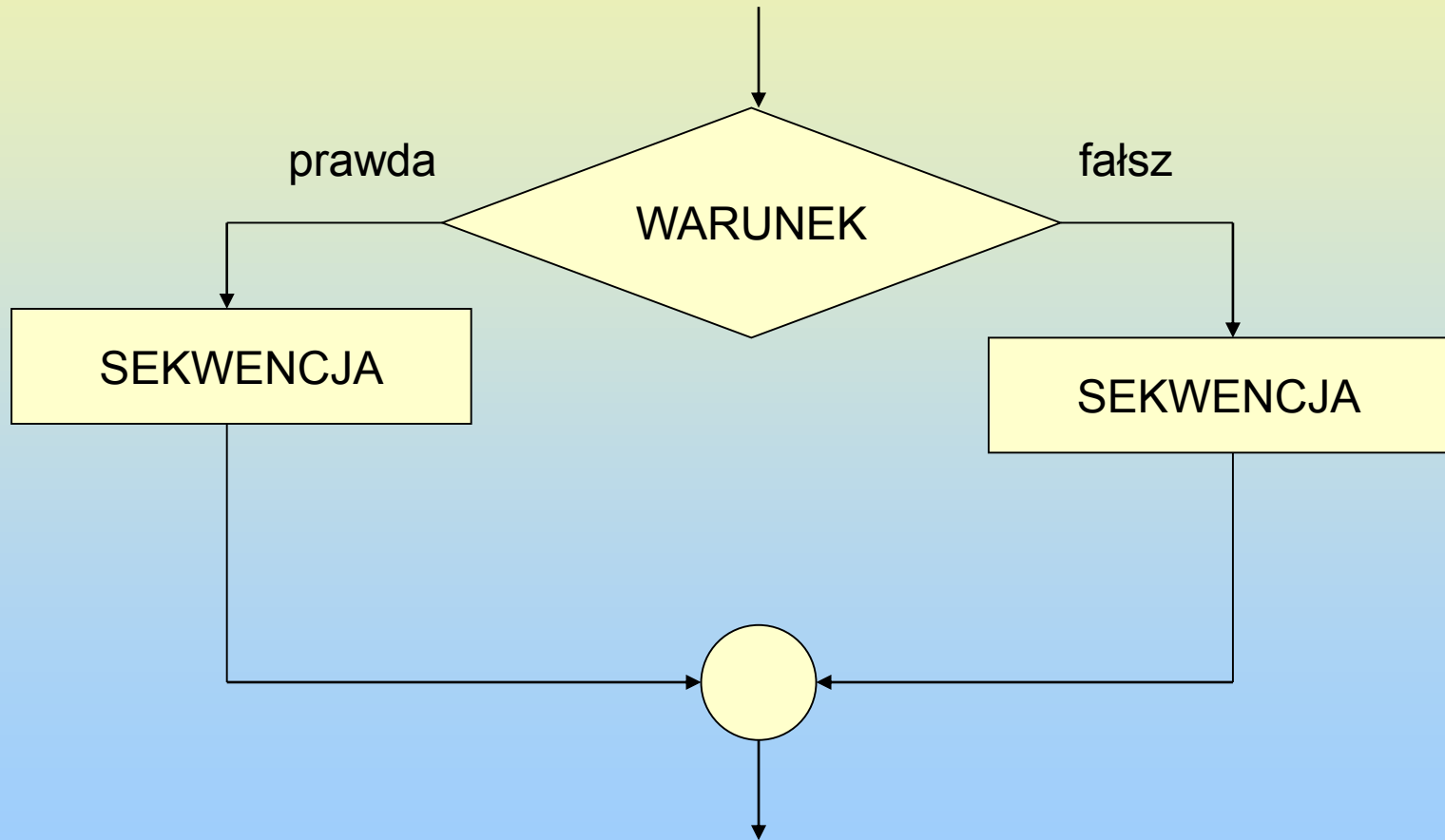
```
nazwa := nazwa || ' bazy danych';
```

```
dbms_output.put_line(nazwa);
```

```
END;
```

Rodzaje struktur sterujących

SELEKCJA



SELEKCJA - instrukcja IF ... THEN

postać podstawowa

```
IF warunek THEN  
    sekwencja poleceń 1  
END IF;
```

postać rozszerzona

```
IF warunek THEN  
    sekwencja poleceń 1  
ELSE  
    sekwencja poleceń 2  
END IF;
```

postać pełna

```
IF warunek 1 THEN  
    sekwencja poleceń 1  
ELSIF warunek 2 THEN  
    sekwencja poleceń 2  
ELSIF warunek 3 THEN  
    sekwencja poleceń 3  
...  
ELSE  
    sekwencja poleceń n  
END IF;
```

SELEKCJA - instrukcja IF ... THEN

Przykład

```
DECLARE  
prawda BOOLEAN := true;  
BEGIN  
IF prawda THEN  
  dbms_output.put_line('prawda');  
ELSE  
  dbms_output.put_line('fałsz');  
END IF;  
END;
```

SELEKCJA - instrukcja CASE

postać prosta

```
CASE wyrażenie  
WHEN wartość 1 THEN  
sekwencja poleceń 1  
WHEN wartość 2 THEN  
sekwencja poleceń 2  
...  
[ELSE  
sekwencja poleceń n]  
END [CASE];
```

postać z listą wyrażeń

```
CASE  
WHEN warunek 1 THEN  
sekwencja poleceń 1  
WHEN warunek 2 THEN  
sekwencja poleceń 2  
...  
[ELSE  
sekwencja poleceń n]  
END [CASE];
```

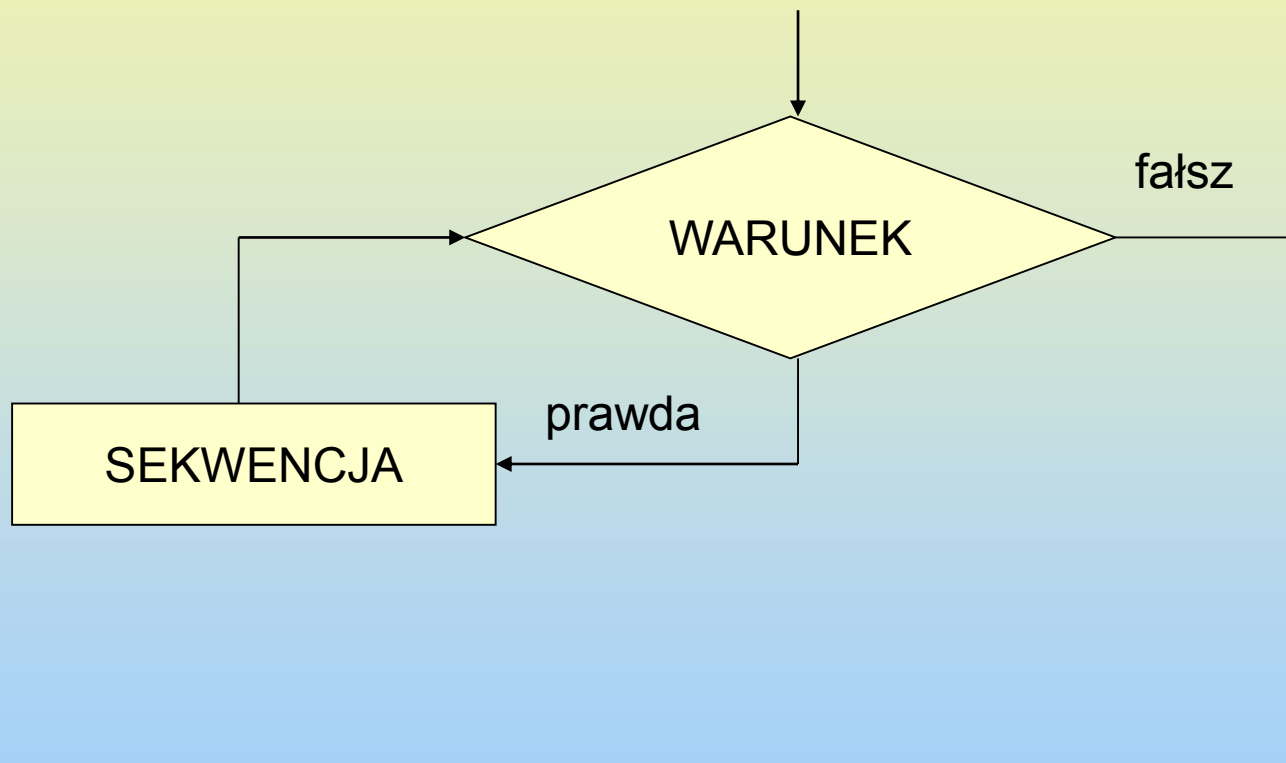
SELEKCJA - instrukcja CASE

Przykład

```
DECLARE  
vat number(2,2) := 0.22;  
proc varchar2(20);  
BEGIN  
proc := CASE vat  
WHEN 0 THEN '0%'  
WHEN 0.7 THEN '7%'  
WHEN 0.22 THEN '22%'  
END;  
dbms_output.put_line(proc);  
END;
```

```
DECLARE  
vat number(2,2) := 0.22;  
proc varchar2(20);  
BEGIN  
CASE  
WHEN vat = 0 THEN  
    proc := '0%';  
WHEN vat = 0.7 THEN  
    proc := '7%';  
WHEN vat = 0.22 THEN  
    proc := '22%';  
END CASE;  
dbms_output.put_line (proc);  
END;
```


Rodzaje struktur sterujących ***ITERACJA***



ITERACJA - instrukcja LOOP

Pętla bezwarunkowa

LOOP

sekwencja poleceń

END LOOP;

Pętla z EXIT WHEN

LOOP

sekwencja poleceń

EXIT WHEN

warunek;

END LOOP;

Pętla z EXIT

LOOP

sekwencja poleceń

IF warunek **THEN**

EXIT;

END IF;

END LOOP;

ITERACJA - instrukcja LOOP

Przykład

DECLARE

licznik number(2) := 0;

liczba_iteracji **CONSTANT** number(2) := 5;

BEGIN

LOOP

licznik := licznik + 1;

dbms_output.put_line('Iteracja nr ' || to_char(licznik));

EXIT WHEN licznik = liczba_iteracji;

END LOOP;

END;

ITERACJA - instrukcja WHILE

Postać

```
WHILE warunek LOOP  
sekwencja poleceń  
END LOOP;
```

Przykład

```
DECLARE  
  licznik NUMBER(2) := 0;  
  liczba_iteracji CONSTANT NUMBER(2) := 5;  
BEGIN  
  WHILE licznik < liczba_iteracji LOOP  
    licznik := licznik + 1;  
    dbms_output.put_line('Iteracja nr ' || to_char(licznik));  
  END LOOP;  
END;
```

ITERACJA - instrukcja FOR

Postać

```
FOR licznik IN [REVERSE] dolna_granica .. górna_granica LOOP  
    sekwencja poleceń  
END LOOP;
```

Przykład

```
DECLARE  
    liczba_iteracji CONSTANT NUMBER(2) := 5;  
BEGIN  
    FOR licznik IN 1.. liczba_iteracji LOOP  
        dbms_output.put_line('Iteracja nr ' || to_char(licznik));  
    END LOOP;  
END;
```

Instrukcja NULL

- Nie wykonuje żadnej akcji
- Umożliwia testowanie struktur sterujących na etapie projektowania programu

Przykład

```
DECLARE  
czy_egzaminzdany BOOLEAN := true;  
BEGIN  
IF NOT czy_egzaminzdany THEN  
NULL;  
ELSE  
dbms_output.put_line('Egzamin zdany!');  
END IF;  
END;
```

ZAPYTANIA w PL/SQL

- Zapytanie musi zwrócić dokładnie jeden rekord
- Zapytanie musi zawierać klauzulę **INTO**, z:
 - listą zmiennych prostych, których liczba musi odpowiadać liczbie wyrażeń w klauzuli **SELECT** zapytania – obowiązuje również zgodność typów
- lub
 - zmienną rekordową o strukturze zgodnej ze strukturą rekordu zwróconego przez zapytanie
- Na końcu polecenia średnik

ZAPYTANIA w PL/SQL

DECLARE

suma NUMBER(6,2);

liczba NUMBER(5);

instytut instytuty%ROWTYPE;

BEGIN

SELECT * INTO instytut **FROM** instytuty

WHERE nazwa = 'INFORMATYKA';

SELECT sum(placa), count(*)

INTO suma, liczba

FROM pracownicy **WHERE** id = instytut.id;

dbms_output.put_line('Suma płac: ' || to_char(suma));

dbms_output.put_line('Pracowników: ' || to_char(liczba));

END;

INSERT, UPDATE, DELETE w PL/SQL

- Postać **INSERT, UPDATE, DELETE** – taka sama jak w SQL
- Opcjonalnie można dodać klauzulę **RETURNING INTO**, która pozwala na zapisanie we wskazanej zmiennej:
 - wartości atrybutów rekordu, wstawionego przez zlecenie **INSERT**
 - wartości atrybutów rekordu, zmodyfikowanego przez zlecenie **UPDATE**
 - wartości atrybutów rekordu, usuniętego przez zlecenie **DELETE**

Zmienne rekordowe - wykorzystanie

Przykład

DECLARE

id pracownicy.id_prac%TYPE;

BEGIN

INSERT INTO pracownicy (id_prac, imie, nazwisko)

VALUES (100, 'Jan','Kowalski')

RETURNING id_prac **INTO** id;

dbms_output.put_line('Identyfikator nowego pracownika: ' ||
to_char(id));

END;

Zmienne rekordowe – INSERT, UPDATE

- Przykład

DECLARE

instytut instytuty%ROWTYPE;

BEGIN

instytut.id := 20;

instytut.nazwa := 'INFORMATYKA';

instytut.adres := 'PODCHORAŹYCH 1';

INSERT INTO instytuty **VALUES** instytut;

instytut.nazwa := INFORMATYKA STOSOWANA';

instytut.adres := 'WARSZAWSKA 24';

UPDATE instytuty **SET ROW** = instytut

WHERE id = 20;

END;